EEL 6764 Graduate Computer Architecture
Midterm Exam Practice Solutions

April 9, 2025

Short Answer Questions

1

Question:
Some microprocessors today are designed to have adjustable voltage, so 15 % reduction in voltage may result in 15 % reduction in frequency. What is the impact on dynamic power? Show all your work. No credit if supporting work is not shown.

When voltage is reduced by 15% and frequency is reduced by 15%, we need to calculate the impact on dynamic power.

Using the power formula: $P \propto \frac{1}{2} \times C \times V^2 \times f$

With $V' = 0.85V$ and $f' = 0.85f$:

$$\begin{aligned} P' &= \frac{1}{2} \times C \times (0.85V)^2 \times (0.85f) && (1) \\ &= \frac{1}{2} \times C \times (0.7225V^2) \times (0.85f) && (2) \\ &= 0.614 \times \left(\frac{1}{2} \times C \times V^2 \times f\right) && (3) \\ &= 0.614 \times P && (4) \end{aligned}$$

Therefore, the dynamic power is reduced to approximately 61.4% of the original power, representing a 38.6% reduction in power consumption.

2. Higher associativity reduces miss rate

True. Higher associativity generally reduces miss rate by decreasing conflict misses. However, it comes with trade-offs including longer hit times (due to more comparisons needed), higher power consumption, and increased hardware complexity and cost.

3. Priority of read misses over write misses

We should give priority to read misses because they directly block processor execution since instructions need the data to continue computations, while write misses can be handled asynchronously through write buffers without stalling the processor, making read misses a bigger priority for maintaining performance.

4. Amdahl's Law application for processor speedup

Given:

- 30% time on data movement (loads/stores)
- 40% time on ALU operations
- 20% time on control flow

- 10% time on other operations
- Only ALU operations can be sped up
- Target speedup: 8x

Using Amdahl's Law:

$$\text{Speedup} = \frac{1}{(1 - F_{\text{enhanced}}) + \frac{F_{\text{enhanced}}}{S_{\text{enhanced}}}} \tag{5}$$

Where $F_{\text{enhanced}} = 0.4$ (ALU operations fraction)

$$8 = \frac{1}{(1 - 0.4) + \frac{0.4}{S_{\text{enhanced}}}} \tag{6}$$

$$8 = \frac{1}{0.6 + \frac{0.4}{S_{\text{enhanced}}}} \tag{7}$$

$$8\left(0.6 + \frac{0.4}{S_{\text{enhanced}}}\right) = 1 \tag{8}$$

$$4.8 + \frac{3.2}{S_{\text{enhanced}}} = 1 \tag{9}$$

$$\frac{3.2}{S_{\text{enhanced}}} = 1 - 4.8 = -3.8 \tag{10}$$

Since we have a negative value, and speedup cannot be negative, this is mathematically impossible. No matter how much you speed up the ALU operations, you cannot achieve an 8x overall speedup with only 40% of the execution time being enhanced.

According to Amdahl's Law, the maximum theoretical speedup would be:

$$\text{Speedup}_{\text{max}} = \frac{1}{1 - 0.4} = \frac{1}{0.6} = 1.67\text{x} \tag{11}$$

5. Execution time as measure of performance

False. While execution time is important, it alone is not a sufficient measure of computer performance. Other factors to consider include:

- Power/energy efficiency
- Throughput (tasks completed per unit time)
- Scalability
- System responsiveness
- Memory usage
- Cost-performance ratio
- Real-time constraints

Modern performance evaluation requires a balanced consideration of multiple metrics.

6. Data forwarding cannot resolve data hazard

Data forwarding cannot resolve hazards in several cases:

1. **Load-use hazards**: When an instruction depends on data being loaded from memory by a previous instruction, but the data isn't available in time. For example:

   ```
   LW $t0, 0($s0)    # Load value from memory
   ADD $t1, $t0, $t2 # Use the loaded value immediately
   ```

   The value in $t0 isn't available until the memory stage, but is needed in the execute stage.

2. **Long-latency operations**: When operations like floating point division take multiple cycles to complete, but results are needed sooner.

3. **Multi-cycle dependencies**: When there are dependencies spanning multiple instructions that cannot be resolved by simple forwarding.

7. Write-through policy usage

False. In today's processors, write-back policy is much more widely used than write-through.

Reasons why write-back is preferred:

- It reduces memory traffic by only writing modified cache lines when necessary
- It significantly lowers power consumption by reducing off-chip communication
- With large caches and increasing memory latency gaps, write-back is more efficient
- Write-back policy works better with write buffers to hide latency

Write-through is mainly used in specialized systems where data consistency is critical or in some embedded systems with simpler memory hierarchies.

8. Branch prediction accuracy

Given:

- Branches comprise 20% of all instructions
- 2-cycle stall on each misprediction
- Target average branch penalty: no more than 30% of ideal CPI

Let's calculate the required branch prediction accuracy:
Branch penalty = BF $\times$ (1 - BPA) $\times$ SP
Where:

- BF = Branch Frequency = 20% = 0.2
- BPA = Branch Prediction Accuracy (what we're solving for)
- SP = Stall Penalty = 2 cycles

Target branch penalty = 30% of ideal CPI = 0.3
So:

$$\begin{aligned} 0.3 &= 0.2 \times (1 - \text{BPA}) \times 2 && (12) \\ 0.3 &= 0.4 \times (1 - \text{BPA}) && (13) \\ 0.75 &= 1 - \text{BPA} && (14) \\ \text{BPA} &= 0.25 && (15) \end{aligned}$$

Therefore, the branch prediction accuracy must be at least 75% to keep the branch penalty below 30% of the ideal CPI.

9. Fetching more than one block on a miss

In **prefetching** scheme, on a miss, we fetch more than one block to reduce miss penalty or miss rate. The prefetching can be hardware-based (automatically detecting access patterns) or software-based (compiler-inserted prefetch instructions).

10. Way-predicting caches characteristics

For way-predicting caches:

- Bandwidth is **better**
- Power consumption is **better**
- Hardware cost is **high**

2. Multiple Choice Questions

1. Popular page replacement algorithm in processors: **(b) LRU (Least Recently Used)**
2. Hazards in an out-of-order processor: **(a) RAW, (b) WAW, (c) WAR**
3. Number of dies per 300mm wafer for a 1.5cm die: **(c) 270** (calculated using the formula: Dies = $\pi(\text{wafer diameter}/2)^2/\text{die area} - \pi \times \text{wafer diameter}/\sqrt{2 \times \text{die area}}$)
4. Multi-level cache goal: **(a) reduce L1 hit time, (b) reduce L1 miss rate**
5. Branch prediction schemes that perform better than 1-bit: **(a) 2-bit, (c) Correlating, (d) All of the Above**

6. Advantages of virtual memory: **(a) User does not have to worry about actual physical memory size, (b) User is given an illusion of an infinite memory**

7. Supply voltage scaling helps in improving: **(a) Power, (b) Energy**

8. Write Back scheme can be improved by: **(a) Write Buffer, (b) Critical Word first**

9. Cache performance improvement techniques: **(a) Larger block size, (b) Higher Associativity, (c) Multi-level Cache**

10. Main advantages of merging write buffer: **(b) reduces miss penalty**

3. Memory Hierarchy

Advanced Cache Optimization Techniques

a) Nonblocking cache

Allows the processor to continue executing instructions and making cache accesses even when there's a cache miss being processed. This technique increases cache bandwidth by supporting "hit under miss" capability, where cache hits can be serviced while a miss is being resolved. This improves performance by overlapping computation with memory access time, effectively hiding part of the miss penalty.

b) Way predicting cache

Adds extra bits to predict which way in a set-associative cache is likely to contain the requested data. The multiplexor is set early to select the predicted block, and only one tag comparison is performed initially. If the prediction is wrong, the other ways are checked in subsequent cycles. This reduces both hit time and power consumption for correct predictions, at the cost of additional complexity.

c) Critical word first

Prioritizes fetching the specific word requested by the processor from memory before retrieving the rest of the cache line. Once the critical word arrives, it's forwarded to the processor immediately, allowing execution to resume while the remainder of the cache line is being filled. This technique reduces the effective miss penalty by enabling the processor to continue operation sooner.

d) Hardware prefetching

Proactively fetches data into the cache before it's explicitly requested by analyzing memory access patterns. The prefetcher can bring in sequential cache lines (stream prefetching) or follow more complex patterns using prediction tables. This technique reduces miss rates by anticipating future memory accesses and hiding memory latency.

Critical Word First and Early Restart Calculation

Given:

- 1 MB L2 cache with 64-byte blocks
- Refill path is 16 bytes wide
- L2 can be written with 16 bytes every 4 processor cycles
- Time to receive first 16-byte block from memory controller: 120 cycles
- Each additional 16-byte block requires 16 cycles
- Data can be bypassed into L2 cache read port

Without critical word first/early restart:

- The entire 64-byte block must be transferred before the processor can continue
- Total cycles = $120 + (3 \times 16) = 120 + 48 = 168$ cycles

With critical word first/early restart:

- Assuming the critical word is in the first 16-byte block
- Processor can continue after receiving the first block
- Total cycles = 120 cycles

The difference represents a significant reduction in effective miss penalty.

4. Technology Trends, Performance Measurement

1. ISA Types

Stack ISA

Uses a stack-based approach where operations are performed on the top elements of the stack. Operands are implicitly accessed from the stack, reducing instruction size but potentially increasing the number of instructions needed. Examples include Java Virtual Machine bytecode.

Accumulator ISA

Uses a single register (accumulator) as an implicit operand for most instructions. Results of operations are stored in the accumulator. This design simplifies instruction encoding but may require more memory accesses. Early computers like EDSAC used this approach.

Register-memory ISA

Allows operations between registers and memory, where one operand can be in memory and another in a register. This increases flexibility but makes instruction encoding more complex. Examples include x86 architecture.

Register-register ISA

Restricts most operations to using registers only. Memory access is limited to load and store instructions. This leads to simpler instruction execution but may require more instructions for complex operations. RISC architectures like MIPS and RISC-V use this approach.

2. Clock Cycle and Speedup Calculations

Given:

- Original single-cycle implementation: 13 ns cycle time
- Stage times: IF = 2 ns, ID = 5 ns, EX = 1 ns, MEM = 4 ns, WB = 2.5 ns
- Pipeline register delay = 0.2 ns
- Stall every 4 instructions

a) Clock cycle time of 5-stage pipeline

Clock cycle time = Maximum stage time + Pipeline register delay = 5 ns + 0.2 ns = 5.2 ns

b) CPI with stalls

Ideal CPI = 1
Stall cycles per instruction = 1 stall / 4 instructions = 0.25
CPI = 1 + 0.25 = 1.25

c) Speedup of pipelined vs. single-cycle

$$\begin{aligned}
\text{Speedup} &= \frac{\text{Single-cycle time}}{\text{Pipelined cycle time} \times \text{CPI}} && (16)\\
&= \frac{13\text{ ns}}{5.2\text{ ns} \times 1.25} && (17)\\
&= \frac{13}{6.5} && (18)\\
&= 2 && (19)
\end{aligned}$$

d) Speedup with infinite pipeline stages

With infinite stages, each stage would approach zero time.
The limiting factor becomes the pipeline register delay (0.2 ns).

$$\begin{aligned}\text{Speedup} &= \frac{\text{Single-cycle time}}{\text{Pipeline register delay}} && (20)\\ &= \frac{13\text{ ns}}{0.2\text{ ns}} && (21)\\ &= 65 && (22)\end{aligned}$$

5. Basic Pipelining and RISC-V Architecture

1. 2-bit Dynamic Branch Prediction

A 2-bit dynamic prediction scheme uses a 2-bit saturating counter to track branch behavior, with four states typically labeled:

- Strongly not taken (00)
- Weakly not taken (01)
- Weakly taken (10)
- Strongly taken (11)

The state transitions based on branch outcomes. For example, if the current state is "weakly taken" and a branch is actually taken, the state moves to "strongly taken."

This scheme performs better than static prediction because:

1. It adapts to runtime behavior rather than using compile-time heuristics
2. It provides hysteresis, requiring multiple mispredictions to change the prediction direction
3. It handles irregular but biased branch patterns better

The scheme works well for loop branches (taken many times, then not taken once) and other patterns with strong bias but occasional deviations.

2. Better than 2-bit Prediction

Yes, we can do better than 2-bit prediction with:

1. **Correlating predictors**: Use the history of recent branches to predict the next branch, capturing patterns where branches are correlated.
2. **Tournament predictors**: Combine multiple prediction schemes and select the one that's working best.

3. **Neural branch predictors**: Use neural network techniques to learn complex branch patterns.

4. **TAGE predictors**: Use multiple predictor tables indexed with different history lengths.

5. **Perceptron predictors**: Apply machine learning algorithms to branch prediction.

These advanced schemes can significantly reduce misprediction rates, particularly for branches with complex patterns.

3. RISC-V Base ISA Architecture

a) Four ISA design principles of RISC-V

1. **Simplicity**: Clean, minimal instruction set with regularized encoding

2. **Modularity**: Base ISA with optional standard extensions

3. **Extensibility**: Reserved opcode space for custom extensions

4. **Stability**: Base ISA and standard extensions are frozen, ensuring long-term compatibility

b) Number of registers

- Integer registers: 32

- FP registers: 32

c) Instruction word length

32 bits

d) Number of instruction formats

6 (R, I, S, B, U, J)